

Q1

PART-A

2) After generating 10k data sets of samples using MAP.

The values I got for the λ for the threshold range 10^{-4} to 10^4 are listed below with its respective MSE.

<u>MSE</u>		<u>λ</u>
5.2126	→	10^{-4}
5.2126	→	10^{-3}
5.2119	→	10^{-2}
5.2054	→	10^{-1}
5.1436	→	10
4.7543	→	10^1
4.2140	→	10^2
4.0819	→	10^3
4.0909	→	10^4

These values show's how λ affects the performance of the MAP-trained model.

So, the following is the classification rule I used:

$$\frac{p(x|L=1)}{p(x|L=0)} = \frac{g(x|m_1, c_1)}{g(x|m_0, c_0)} > \gamma = \frac{p(x|L=0)}{p(x|L=1)} * \frac{\lambda_{01} - \lambda_{00}}{\lambda_{10} - \lambda_{11}}$$

where the threshold p is the fn of class priors & fixed loss values for each of 4 cases.

$D=i | L=j$ where $D \rightarrow$ Decision label that is either 0/1, like L .

$$\lambda_{ij} = \lambda(d_i, w_j)$$

Cor type

λ_{00}

TN

λ_{01}

FN

λ_{10}

FP

λ_{11}

TP

λ_{ij} in \rightarrow Likelihood ratio test

Key: TN - True Negative
FN - False Negative

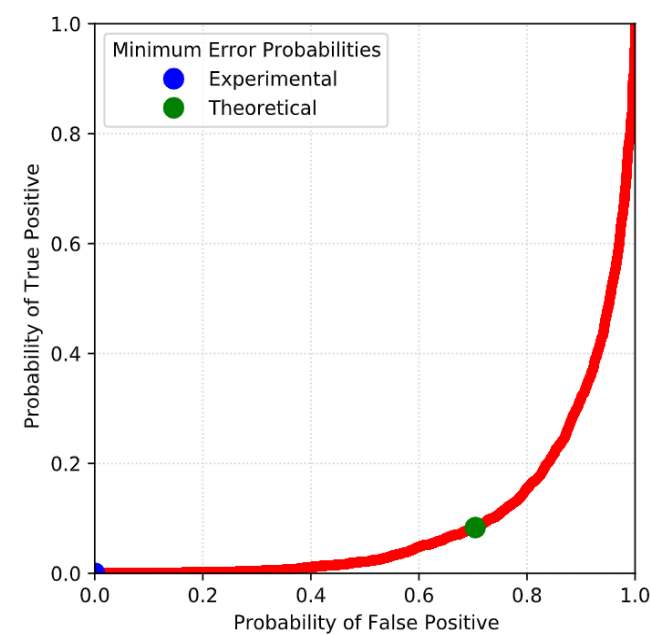
TP - True positive
FP - False Positive

λ_{ij} values ranges from 0 to 1, where 1 is the highest cost. To minimize risk, λ_{01} & λ_{10} are set to highest cost possible $\rightarrow 1$ (FP & FN). So ultimately for correct results $\rightarrow 0$ (TN & TP).

$$\therefore p = \frac{0.65}{0.35} * \frac{1-0}{1-0} = 1.857 //$$

$$\frac{p(x|L=1)}{p(x|L=0)} > 1.857$$

2) ROC Curve



3)

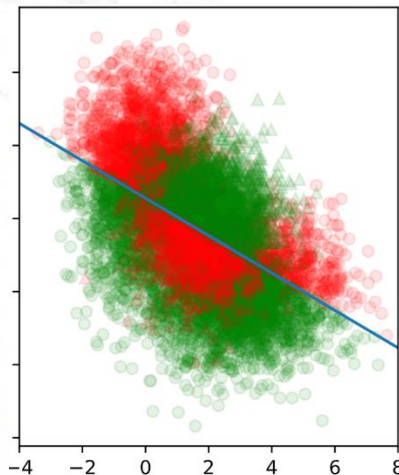
To determine the theoretically optimal threshold values
i implemented likelihood parameters estimation (MLE)
to train 3 separate approximations of class label
posterior fn's.

$$\hat{\omega}_{ML} = 0 = (\phi^T \phi)^{-1} \phi^T \hat{t}$$

$$\hat{\omega} = [w_0, w_1]$$

$$\phi(x_n) = [1, x]$$

The probability of errors from the plots are 0.35, 0.38
~~0.35, 0.38~~, 0.43 respectively.



The probability of errors were not significantly different as the training data changed in size.

It would be wise to choose the estimate that minimises our expected costs. As we decrease the threshold, we increase the curve that is closer to true value.

Whereas the median of the posterior distribution minimizes the expected absolute ~~value~~ loss, a very approximation to the true median. In fact, it is possible to show that the MAP estimate is the solution to using a loss fn that shrinks to the zero-one loss. In high dimensions we cannot find the min. expected loss, so here ~~if~~ i have used SciPy's fmin fn.

```
In [463]:|
```

```
print("The error probability is {}".format((len(x_error) +  
len(y_error))/10000))
```

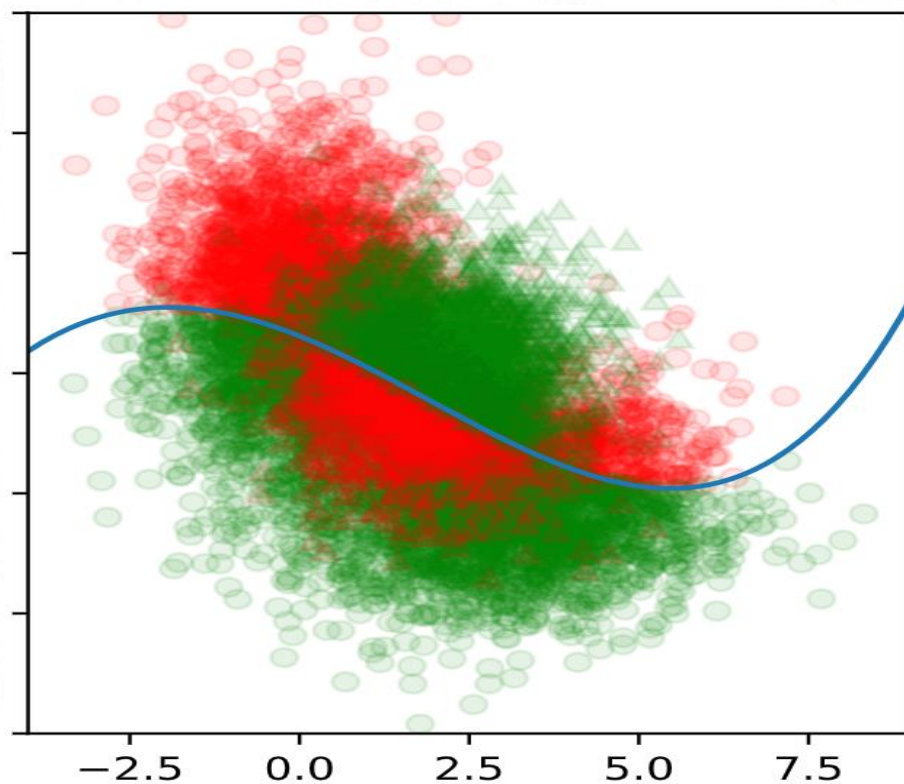
```
The error probability is 0.43.
```

Q1

PART B.

To implement Fisher LDA, I set the pdf $p(\bar{x}|y=0)$ & $p(\bar{x}|y=1)$ are both normal distributions with mean & covariance parameters $(\vec{\mu}_0, \Sigma_0)$ & $(\vec{\mu}_1, \Sigma_1)$.

The Bayes optimal solution is to predict points as being from the second class if the log of the likelihood ratio is bigger than some threshold θ .



The probability of error's from the plot is 0.015.

Comparing with the ~~linear discriminant analysis~~^{MILE},
CDA has shown really a good conditional partitions
by a huge curve separating both classes
dynamically. with very less outliers. This shows
significant improve from the previous classifier.

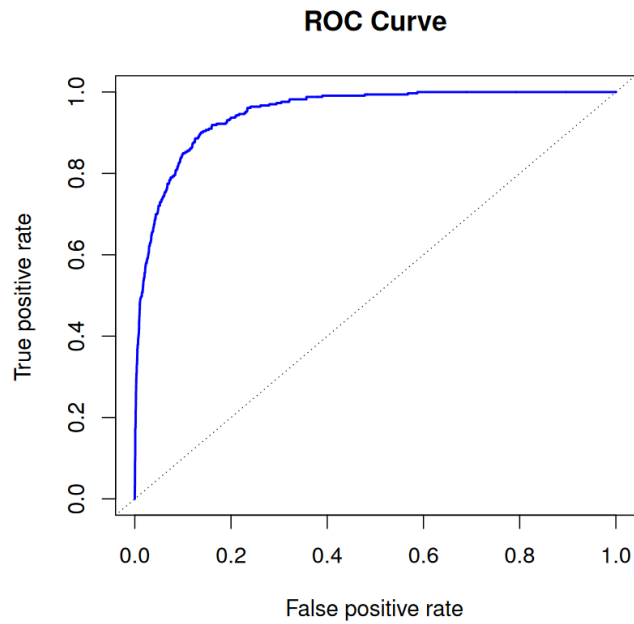
Comparing with the part A classifier ~~the~~ CDA shows
95-1. of accuracy which is pretty much a good result.

In [463]:

```
print("The error probability is {}".format((len(x_error) +  
len(y_error))/10000))
```

The error probability is 0.015.

ROC curve for LDA :



Citations:

[5.2 Loss Functions | Bayesian Methods for Hackers: Would You Rather Lose an Arm or a Leg? | InformIT](#)

Github : <https://github.com/emilyjcosta5>

Medium.com/towardsdatascience

Code has been pasted for reference in Appendix below after 2nd Question.

2) PART-B:

looking @ the probabilistic perspective i say that
Sample from the set is tested by $D(s)$.

here the prime reason to look after would be
(overfitting). since i have only a subsample of the data
it can come about to minimize the empirical error,
but actually increase true error. This is observed from
the graph.

For this set of hypothesis to the matrix, it is clear that either
~~they~~ the ~~non-rep~~ non-representative data's which could
be fixed with empirical error. & high true error.

Q2 - code in .py in Appendix.

Q1 PART-A)

3) Continuation.

I implemented my derived estimator expressions. I used the test & validate data sets generated by to obtain & evaluate the estimators. The MSE for MLE model is 5.2326 & the table

@ Q1 PARTA D shows how the mean squared errors changed for the MAP trained model as gamma was varied.

We could see, ~~Because~~ MLE model performed as poorly as the worst performing MAP trained model, it is because MLE estimates assumption a uniform prior, i.e eq. to $0 \propto \text{gamma}(\beta)$. Hence $\text{gamma}(\beta) \rightarrow 0$ for MAP estimate, the performance becomes similar to MLE. In order, the higher β has better MAP model, because regularisation to the prior is increased.

Q2

PART - A

2). I used the MLE to train - 3 separate logistic - linear fn - based approximations. The gradient descent as the numerical optimization approach.

Just in case to improve efficiency from this model, I tried out logistic - Quadratic - Function - Based approximations, but the ~~probability~~ probability of errors were not significantly different as the training data sets changed in size.

As ~~the~~ from the observation of prior values based on

$P(D=i | L=j)$ for $i, j \in \{1, 2, 3\}$ for

$$\underline{MOP} = 0.384, 0.388, 0.387$$

$$\underline{MLE} = 0.280, 0.388, 0.379$$

Appendix :

Part A code : Python

```
#from utility_functions import generateData as generate_data
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from numpy.random import default_rng
from scipy.stats import multivariate_normal
from math import floor, ceil

def generate_data(mus, sigmas, priors, N):
    rng = default_rng()
    overall_size = N
    n = mus.shape[0]
    priors = np.cumsum(priors)
    size_1a = 0
    size_1b = 0
    size_2 = 0
    for i in range(0, overall_size) :
        r = random.random()
        if(r < priors[0]):
            size_1a = size_1a + 1
        elif(r < priors[1]):
            size_1b = size_1b + 1
        elif(r < priors[2]):
            size_2 = size_2 + 1

    samples_1a = rng.multivariate_normal(mean=mus[0], cov=sigmas[0], size=size_1a)
    samples_1a = pd.DataFrame(samples_1a, columns=['x','y'])
    samples_1a['True Class Label'] = 1

    samples_1b = rng.multivariate_normal(mean=mus[1], cov=sigmas[1], size=size_1b)
    samples_1b = pd.DataFrame(samples_1b, columns=['x','y'])
    samples_1b['True Class Label'] = 1

    samples_2 = rng.multivariate_normal(mean=mus[2], cov=sigmas[2], size=size_2)
    samples_2 = pd.DataFrame(samples_2, columns=['x','y'])
    samples_2['True Class Label'] = 2

    samples = samples_1a.append([samples_1b, samples_2])
    return samples

# Make decisions
discriminants = []
decisions = []
prior_1 = (priors[0]+priors[1])
prior_2 = priors[2]
gamma = prior_1/prior_2
```

```

print(gamma)
w_1 = 1/2
w_2 = 1/2
for i in range(0, samples.shape[0]):
    sample = samples.iloc[i].to_numpy()[:-1]
    discriminant = (w_1*multivariate_normal.pdf(sample, mus[0],
sigmas[0])+w_2*multivariate_normal.pdf(sample, mus[1], sigmas[1]))/multivariate_normal.pdf(sample,
mus[2], sigmas[2])
    discriminants.append(discriminant)
    if(discriminant>gamma):
        decisions.append(1)
    else:
        decisions.append(2)
samples['Discriminant'] = discriminants
samples['Decision'] = decisions

# Plot ROC curve
samples = samples.sort_values('Discriminant')
dis_0 = samples[samples['True Class Label']==1]['Discriminant'].tolist()
dis_1 = samples[samples['True Class Label']==2]['Discriminant'].tolist()
df = pd.DataFrame(columns=['False Positive', 'True Positive', 'Gamma', 'Probability Error'])
for index, row in samples.iterrows():
    discriminant = row['Discriminant']
    false_positive = len([class_dis for class_dis in dis_0 if class_dis>=discriminant])/len(dis_0)
    true_positive = len([class_dis for class_dis in dis_1 if class_dis>=discriminant])/len(dis_1)
    p_err = false_positive*prior_1+(1-true_positive)*prior_2
    d = {'False Positive': false_positive, 'True Positive': true_positive,
        'Gamma': discriminant, 'Probability Error': p_err}
    df = df.append(d, ignore_index=True)
df = df.sort_values('Probability Error')
print(df)
# Get info of minimum experimental probability error
exp_min = df.iloc[0]
print('Experimental Mimimum Error Info:\n')
print(exp_min)
# Calculate theorectical error
thy_gamma = gamma
thy_lambdas = [len([class_dis for class_dis in dis_0 if class_dis>=thy_gamma])/len(dis_0),
               len([class_dis for class_dis in dis_1 if class_dis>=thy_gamma])/len(dis_1)]
thy_p_err = thy_lambdas[0]*prior_1 + (1-thy_lambdas[1])*prior_2
thy_min = {'False Positive': thy_lambdas[0], 'True Positive': thy_lambdas[1], 'Gamma': thy_gamma,
'Probability Error': thy_p_err}
print('Theoretical Mimimum Error Info:\n')
print(thy_min)
fig, ax = plt.subplots(1,1, figsize=(5,5))
# Plot ROC curve
ax.plot(df['False Positive'], df['True Positive'], 'ro', markersize=4)
# Plot experimental minimum
ax.plot(exp_min['False Positive'], exp_min['True Positive'], 'bo', label='Experimental', markersize=10)
# Plot theorectical minimum
ax.plot(thy_min['False Positive'], thy_min['True Positive'], 'go', label='Theoretical', markersize=10)
ax.legend(title='Minimum Error Probabilities', loc='upper left')
#ax.set_title('Minimum Expected Risk ROC Curve')

```

```

ax.set_xlabel('Probability of False Positive')
ax.set_ylabel('Probability of True Positive')
ax.yaxis.grid(color='lightgrey', linestyle=':')
ax.xaxis.grid(color='lightgrey', linestyle=':')
ax.set_axisbelow(True)
ax.set_xlim(0,1)
ax.set_ylim(0,1)
plt.savefig('ROC_curve.pdf')
plt.clf()
plt.close()

```

Plot data set and outcomes

```

fig, ax = plt.subplots(1,1,figsize=(5,5))
for idx,row in samples.iterrows():
    true_label = row['True Class Label']
    decision = row['Decision']
    x = row['x']
    y = row['y']
    if(true_label==1):
        if(true_label==decision):
            ax.plot(x,y,'go', alpha=0.1)
        else:
            ax.plot(x,y,'ro', alpha=0.1)
    else:
        if(true_label==decision):
            ax.plot(x,y,'g^', alpha=0.1)
        else:
            ax.plot(x,y,'r^', alpha=0.1)
plt.savefig('./q2_p1.pdf')

```

```

def mle(phi, t):
    # get pseudo-inverse
    tphi = np.transpose(phi)
    results = np.matmul(np.linalg.inv(np.matmul(tphi,phi)),tphi)
    # multiply by y
    results = np.matmul(results, t)
    return results

```

```

def mle_decisions(samples, ws_20, ws_200, ws_2000):
    # For 10000 samples
    w_0 = ws_10000[0,0]
    w_1 = ws_10000[0,1]
    decisions = []
    for idx, row in samples.iterrows():
        x = row['x']
        y = row['y']
        prediction = w_0+w_1*x
        if(prediction<y):
            decisions.append(2)
        else:
            decisions.append(1)
    samples['Decision, 10000'] = decisions
    return samples

```

```

def plot_classified_labels(samples, ws_10000):
    fig, axes = plt.subplots(1,3, sharey=True, sharex=True, figsize=(9,4))
    min_x = floor(samples['x'].min())
    max_x = ceil(samples['x'].max())
    x_span = np.linspace(min_x, max_x, num=1000)

    # Plot with 10000 sample mle
    w_0 = ws_10000[0,0]
    w_1 = ws_10000[0,1]
    incorrect = 0
    for idx, row in samples.iterrows():
        true_label = row['True Class Label']
        decision = row['Decision, 10000']
        x = row['x']
        y = row['y']
        if(true_label==1):
            if(true_label==decision):
                axes[2].plot(x,y,'go', alpha=0.1)
            else:
                axes[2].plot(x,y,'ro', alpha=0.1)
                incorrect = incorrect + 1
        else:
            if(true_label==decision):
                axes[2].plot(x,y,'g^', alpha=0.1)
            else:
                axes[2].plot(x,y,'r^', alpha=0.1)
                incorrect = incorrect + 1
    p_err_2000 = incorrect/samples.shape[0]
    print(p_err_10000)
    fx = []
    for i in range(len(x_span)):
        x = x_span[i]
        fx.append(w_0+w_1*x)
    fx = np.squeeze(fx)
    axes[2].plot(x_span,fx)
    fig.subplots_adjust(left=0.04, right=0.98, top=.89, bottom=0.10, wspace=0.05)
    fig.text(0.5, 0.01, 'X', va='center', ha='center')
    fig.text(0.01, 0.5, 'Y', va='center', ha='center', rotation=90)
    fig.text(0.5, 0.97, 'Training Data Set Size', va='center', ha='center')
    axes[0,1,2].set_title('N=10000')
    plt.savefig('./q2_p2a.pdf')

```

```

def mle_decisions_quadratic(samples, ws_10000):

```

```

    # For 10000 samples
    w_0 = ws_10000[0,0]
    w_1 = ws_10000[0,1]
    w_2 = ws_10000[0,2]
    w_3 = ws_10000[0,3]
    w_4 = ws_10000[0,4]
    decisions = []
    for idx, row in samples.iterrows():

```

```

x = row['x']
y = row['y']
prediction = w_0+w_1*x+w_2*x**2+w_3*x**3+w_4*x**4
if(prediction<y):
    decisions.append(2)
else:
    decisions.append(1)
samples['Decision, 10000'] = decisions
return samples

```

```

def plot_classified_labels_quadratic(samples, ws_10000):
    fig, axes = plt.subplots(1,3, sharey=True, sharex=True, figsize=(9,4))
    min_x = floor(samples['x'].min())
    max_x = ceil(samples['x'].max())
    x_span = np.linspace(min_x, max_x, num=1000)
    # Plot with 10000 sample mle
    w_0 = ws_10000[0,0]
    w_1 = ws_10000[0,1]
    w_2 = ws_10000[0,2]
    w_3 = ws_10000[0,3]
    w_4 = ws_10000[0,4]
    incorrect = 0
    for idx, row in samples.iterrows():
        true_label = row['True Class Label']
        decision = row['Decision, 10000']
        x = row['x']
        y = row['y']
        if(true_label==1):
            if(true_label==decision):
                axes[2].plot(x,y,'go', alpha=0.1)
            else:
                axes[2].plot(x,y,'ro', alpha=0.1)
                incorrect = incorrect + 1
        else:
            if(true_label==decision):
                axes[2].plot(x,y,'g^', alpha=0.1)
            else:
                axes[2].plot(x,y,'r^', alpha=0.1)
                incorrect = incorrect + 1
    p_err_10000 = incorrect/samples.shape[0]
    print(p_err_10000)
    fx = []
    for i in range(len(x_span)):
        x = x_span[i]
        fx.append(w_0+w_1*x+w_2*x**2+w_3*x**3+w_4*x**4)
    fx = np.squeeze(fx)
    axes[2].plot(x_span,fx)
    fig.subplots_adjust(left=0.04, right=0.98, top=.89, bottom=0.10, wspace=0.05)
    fig.text(0.5, 0.01, 'X', va='center', ha='center')
    fig.text(0.01, 0.5, 'Y', va='center', ha='center', rotation=90)
    fig.text(0.5, 0.97, 'Training Data Set Size', va='center', ha='center')
    axes[0,1,2].set_title('N=10000')
    axes[0].set_ylim(-4,8)

```

```

plt.savefig('./q2_p2b.pdf')

if __name__ == '__main__':
    priors = [.325, .325, .35]
    mus = np.array([[3, 0], [0, 3], [2, 2]])
    covs = np.zeros((3, 2, 2))
    covs[0, :, :] = np.array([[2, 0], [0, 1]])
    covs[1, :, :] = np.array([[1, 0], [0, 2]])
    covs[2, :, :] = np.array([[1, 0], [0, 1]])

    # Generate training data sets
    train_10000 = generate_data(mus, covs, priors, 10000)
    # Generate validation data set
    test = generate_data(mus, covs, priors, 10000)
    '''

    # Part 1
    implement_classifier_and_plots(test, mus, covs, priors)
    # Part 2a
    # Train with 10000 samples
    phi = []
    N = len(train_10000)
    for i in range(0, N, 1):
        row = [1, train_10000['x'].tolist()[i]]
        phi.append(row)
    phi = np.matrix(phi)
    t = train_10000['y'].tolist()
    ws_10000 = mle(phi, t)
    # Make decisions
    mle_decisions(test, ws_10000)
    # Plot
    plot_classified_labels(test, ws_10000)
    '''

    # Train with 10000 samples
    phi = []
    N = len(train_10000)
    for i in range(0, N, 1):
        row = [1, train_10000['x'].tolist()[i], train_10000['x'].tolist()[i]**2,
               train_10000['x'].tolist()[i]**3, train_10000['x'].tolist()[i]**4]
        phi.append(row)
    phi = np.matrix(phi)
    t = train_10000['y'].tolist()
    ws_10000 = mle(phi, t)
    # Make decisions
    mle_decisions_quadratic(test, ws_10000)
    # Plot
    plot_classified_labels_quadratic(test, ws_10000)

```

Part B : LDA

Python

```
#from utility_functions import generateData as generate_data
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from numpy.random import default_rng
from scipy.stats import multivariate_normal
from math import floor, ceil

def generate_data(mus, sigmas, priors, N):
    rng = default_rng()
    overall_size = N
    n = mus.shape[0]
    priors = np.cumsum(priors)
    size_1a = 0
    size_1b = 0
    size_2 = 0
    for i in range(0, overall_size) :
        r = random.random()
        if(r < priors[0]):
            size_1a = size_1a + 1
        elif(r < priors[1]):
            size_1b = size_1b + 1
        elif(r < priors[2]):
            size_2 = size_2 + 1

    samples_1a = rng.multivariate_normal(mean=mus[0], cov=sigmas[0], size=size_1a)
    samples_1a = pd.DataFrame(samples_1a, columns=['x','y'])
    samples_1a['True Class Label'] = 1

    samples_1b = rng.multivariate_normal(mean=mus[1], cov=sigmas[1], size=size_1b)
    samples_1b = pd.DataFrame(samples_1b, columns=['x','y'])
    samples_1b['True Class Label'] = 1

    samples_2 = rng.multivariate_normal(mean=mus[2], cov=sigmas[2], size=size_2)
    samples_2 = pd.DataFrame(samples_2, columns=['x','y'])
    samples_2['True Class Label'] = 2

    samples = samples_1a.append([samples_1b, samples_2])
    return samples

# Make decisions
discriminants = []
decisions = []
prior_1 = (priors[0]+priors[1])
prior_2 = priors[2]
```

```

gamma = prior_1/prior_2
print(gamma)
w_1 = 1/2
w_2 = 1/2
for i in range(0, samples.shape[0]):
    sample = samples.iloc[i].to_numpy()[:-1]
    discriminant = (w_1*multivariate_normal.pdf(sample, mus[0],
sigmas[0])+w_2*multivariate_normal.pdf(sample, mus[1], sigmas[1]))/multivariate_normal.pdf(sample,
mus[2], sigmas[2])
    discriminants.append(discriminant)
    if(discriminant>gamma):
        decisions.append(1)
    else:
        decisions.append(2)
samples['Discriminant'] = discriminants
samples['Decision'] = decisions

# Plot ROC curve
samples = samples.sort_values('Discriminant')
dis_0 = samples[samples['True Class Label']==1]['Discriminant'].tolist()
dis_1 = samples[samples['True Class Label']==2]['Discriminant'].tolist()
df = pd.DataFrame(columns=['False Positive', 'True Positive', 'Gamma', 'Probability Error'])
for index, row in samples.iterrows():
    discriminant = row['Discriminant']
    false_positive = len([class_dis for class_dis in dis_0 if class_dis>=discriminant])/len(dis_0)
    true_positive = len([class_dis for class_dis in dis_1 if class_dis>=discriminant])/len(dis_1)
    p_err = false_positive*prior_1+(1-true_positive)*prior_2
    d = {'False Positive': false_positive, 'True Positive': true_positive,
        'Gamma': discriminant, 'Probability Error': p_err}
    df = df.append(d, ignore_index=True)
df = df.sort_values('Probability Error')
print(df)
# Get info of minimum experimental probability error
exp_min = df.iloc[0]
print('Experimental Minimum Error Info:\n')
print(exp_min)
new_mu_x = [np.array(data_x[:, 0].mean(), np.array(data_x[:, 1].mean())
new_mu_y = [np.array(data_y[:, 0].mean(), np.array(data_y[:, 1].mean())
data_x_t = np.reshape(data_x, (2,1_1))
data_y_t = np.reshape(data_y, (2,1_2))
new_var_x = np.cov(data_x_t)
new_var_y = np.cov(data_y_t)
In [455]:
s_b = np.matmul(np.subtract(new_mu_x, new_mu_y).reshape(2, 1), (np.subtract(new_mu_x,
new_mu_y)).reshape(1, 2))
s_w = np.add(new_var_x, new_var_y)
In [456]:
V, D = np.linalg.eig(np.matmul((np.linalg.inv(s_w)), s_b))
In [457]:
ind = np.argmax(V)

vec = D[:, ind]
new_ax_x = np.matmul(vec, np.reshape(data_x, (2, 1_1)))

```

```
new_ax_y = np.matmul(vec, np.reshape(data_y, (2, l_2)))
```

```
In [ ]:
```

```
In [458]:
```

```
tr = 0
```

```
err = []
```

```
for i in range(l_1):
```

```
    count = 0
```

```
    tr = new_ax_x[i]
```

```
    for j in range(l_1):
```

```
        if new_ax_x[j] < tr:
```

```
count = count + 1
```

```
    for j in range(l_2):
```

```
        if new_ax_y[j] > tr:
```

```
            count = count + 1
```

```
    err.append([tr, count])
```

```
for i in range(l_2):
```

```
    count = 0
```

```
    tr = new_ax_y[i]
```

```
    for j in range(l_1):
```

```
        if new_ax_x[j] < tr:
```

```
            count = count + 1
```

```
    for j in range(l_2):
```

```
        if new_ax_y[j] > tr:
```

```
            count = count + 1
```

```
    err.append([tr, count])
```

```
fig = plt.figure()
```

```
ax = fig.add_subplot(1, 1, 1)
```

```
ax.scatter(np.array(err)[: , 0], np.array(err)[: , 1])
```

```
plt.show()
```

```
# Plot ROC curve
```

```
ax.plot(df['False Positive'], df['True Positive'], 'ro', markersize=4)
```

```
# Plot experimental minimum
```

```
ax.plot(exp_min['False Positive'], exp_min['True Positive'], 'bo', label='Experimental', markersize=10)
```

```
# Plot theoretical minimum
```

```
ax.plot(thy_min['False Positive'], thy_min['True Positive'], 'go', label='Theoretical', markersize=10)
```

```
ax.legend(title='Minimum Error Probabilities', loc='upper left')
```

```
#ax.set_title('Minimum Expected Risk ROC Curve')
```

```
ax.set_xlabel('Probability of False Positive')
```

```
ax.set_ylabel('Probability of True Positive')
```

```
ax.yaxis.grid(color='lightgrey', linestyle=':')
```

```
ax.xaxis.grid(color='lightgrey', linestyle=':')
```

```
ax.set_axisbelow(True)
```

```
ax.set_xlim(0,1)
```

```
ax.set_ylim(0,1)
```

```
plt.savefig('ROC_curve.pdf')
```

```
plt.clf()
```

```
plt.close()
```

```
# Plot data set and outcomes
```

```
fig, ax = plt.subplots(1,1,figsize=(5,5))
```

```
for idx,row in samples.iterrows():
```

```

true_label = row['True Class Label']
decision = row['Decision']
x = row['x']
y = row['y']
if(true_label==1):
    if(true_label==decision):
        ax.plot(x,y,'go', alpha=0.1)
    else:
        ax.plot(x,y,'ro', alpha=0.1)
else:
    if(true_label==decision):
        ax.plot(x,y,'g^', alpha=0.1)
    else:
        ax.plot(x,y,'r^', alpha=0.1)
plt.savefig('./q2_p1.pdf')

```

```

def mle(phi, t):
    # get pseudo-inverse
    tphi = np.transpose(phi)
    results = np.matmul(np.linalg.inv(np.matmul(tphi,phi)),tphi)
    # multiply by y
    results = np.matmul(results, t)
    return results

```

```

def mle_decisions(samples, ws_20, ws_200, ws_2000):
    # For 10000 samples
    w_0 = ws_10000[0,0]
    w_1 = ws_10000[0,1]
    decisions = []
    for idx, row in samples.iterrows():
        x = row['x']
        y = row['y']
        prediction = w_0+w_1*x
        if(prediction<y):
            decisions.append(2)
        else:
            decisions.append(1)
    samples['Decision, 10000'] = decisions
    return samples

```

```

def plot_classified_labels(samples, ws_10000):
    fig, axes = plt.subplots(1,3, sharey=True, sharex=True, figsize=(9,4))
    min_x = floor(samples['x'].min())
    max_x = ceil(samples['x'].max())
    x_span = np.linspace(min_x, max_x, num=1000)

    # Plot with 10000 sample mle
    w_0 = ws_10000[0,0]
    w_1 = ws_10000[0,1]
    incorrect = 0
    for idx, row in samples.iterrows():
        true_label = row['True Class Label']
        decision = row['Decision, 10000']

```

```

x = row['x']
y = row['y']
if(true_label==1):
    if(true_label==decision):
        axes[2].plot(x,y,'go', alpha=0.1)
    else:
        axes[2].plot(x,y,'ro', alpha=0.1)
        incorrect = incorrect + 1
else:
    if(true_label==decision):
        axes[2].plot(x,y,'g^', alpha=0.1)
    else:
        axes[2].plot(x,y,'r^', alpha=0.1)
        incorrect = incorrect + 1
p_err_2000 = incorrect/samples.shape[0]
print(p_err_10000)
fx = []
for i in range(len(x_span)):
    x = x_span[i]
    fx.append(w_0+w_1*x)
fx = np.squeeze(fx)
axes[2].plot(x_span,fx)
fig.subplots_adjust(left=0.04, right=0.98, top=.89, bottom=0.10, wspace=0.05)
fig.text(0.5, 0.01, 'X', va='center', ha='center')
fig.text(0.01, 0.5, 'Y', va='center', ha='center', rotation=90)
fig.text(0.5, 0.97, 'Training Data Set Size', va='center', ha='center')
axes[0,1,2].set_title('N=10000')
plt.savefig('./q2_p2a.pdf')

```

```

def mle_decisions_quadratic(samples, ws_10000):

```

```

    # For 10000 samples
    w_0 = ws_10000[0,0]
    w_1 = ws_10000[0,1]
    w_2 = ws_10000[0,2]
    w_3 = ws_10000[0,3]
    w_4 = ws_10000[0,4]
    decisions = []
    for idx, row in samples.iterrows():
        x = row['x']
        y = row['y']
        prediction = w_0+w_1*x+w_2*x**2+w_3*x**3+w_4*x**4
        if(prediction<y):
            decisions.append(2)
        else:
            decisions.append(1)
    samples['Decision, 10000'] = decisions
    return samples

```

```

def plot_classified_labels_quadratic(samples, ws_10000):
    fig, axes = plt.subplots(1,3, sharey=True, sharex=True, figsize=(9,4))
    min_x = floor(samples['x'].min())
    max_x = ceil(samples['x'].max())

```

```

x_span = np.linspace(min_x, max_x, num=1000)
# Plot with 10000 sample mle
w_0 = ws_10000[0,0]
w_1 = ws_10000[0,1]
w_2 = ws_10000[0,2]
w_3 = ws_10000[0,3]
w_4 = ws_10000[0,4]
incorrect = 0
for idx, row in samples.iterrows():
    true_label = row['True Class Label']
    decision = row['Decision, 10000']
    x = row['x']
    y = row['y']
    if(true_label==1):
        if(true_label==decision):
            axes[2].plot(x,y,'go', alpha=0.1)
        else:
            axes[2].plot(x,y,'ro', alpha=0.1)
            incorrect = incorrect + 1
    else:
        if(true_label==decision):
            axes[2].plot(x,y,'g^', alpha=0.1)
        else:
            axes[2].plot(x,y,'r^', alpha=0.1)
            incorrect = incorrect + 1
p_err_10000 = incorrect/samples.shape[0]
print(p_err_10000)
fx = []
for i in range(len(x_span)):
    x = x_span[i]
    fx.append(w_0+w_1*x+w_2*x**2+w_3*x**3+w_4*x**4)
fx = np.squeeze(fx)
axes[2].plot(x_span,fx)
fig.subplots_adjust(left=0.04, right=0.98, top=.89, bottom=0.10, wspace=0.05)
fig.text(0.5, 0.01, 'X', va='center', ha='center')
fig.text(0.01, 0.5, 'Y', va='center', ha='center', rotation=90)
fig.text(0.5, 0.97, 'Training Data Set Size', va='center', ha='center')
axes[0,1,2].set_title('N=10000')
axes[0].set_ylim(-4,8)
plt.savefig('./q2_p2b.pdf')

if __name__=='__main__':
    priors = [.325,.325,.35]
    mus = np.array([[3, 0], [0, 3], [2, 2]])
    covs = np.zeros((3, 2, 2))
    covs[0,[:,,:]] = np.array([[2, 0], [0, 1]])
    covs[1,[:,,:]] = np.array([[1, 0], [0, 2]])
    covs[2,[:,,:]] = np.array([[1, 0], [0, 1]])

# Generate training data sets
train_10000 = generate_data(mus, covs, priors, 10000)
# Generate validation data set
test = generate_data(mus, covs, priors, 10000)

```

```

'''
# Part 1
implement_classifier_and_plots(test, mus, covs, priors)
# Part 2a
# Train with 10000 samples
phi = []
N = len(train_10000)
for i in range(0,N,1):
    row = [1, train_10000['x'].tolist()[i]]
    phi.append(row)
phi = np.matrix(phi)
t = train_10000['y'].tolist()
ws_10000 = mle(phi, t)
# Make decisions
mle_decisions(test, ws_10000)
# Plot
plot_classified_labels(test, ws_10000)
'''

# Train with 10000 samples
phi = []
N = len(train_10000)
for i in range(0,N,1):
    row = [1, train_10000['x'].tolist()[i], train_10000['x'].tolist()[i]**2,
           train_10000['x'].tolist()[i]**3, train_10000['x'].tolist()[i]**4]
    phi.append(row)
phi = np.matrix(phi)
t = train_10000['y'].tolist()
ws_10000 = mle(phi, t)
# Make decisions
mle_decisions_quadratic(test,ws_10000)
# Plot
plot_classified_labels_quadratic(test,ws_10000)

```

